EUROPEAN WORKSHOP ON ON-BOARD DATA PROCESING (OBDP2021), 14-17 JUNE 2021

HP4S: HIGH PERFORMANCE PARALLEL PAYLOAD PROCESSING FOR SPACE

Franck Wartel[(1)], Antoine Certain[(2)]

[(1)]Airbus Defence & Space, 31 rue des cosmonautes, 31402 Toulouse, France Email: franck.wartel@airbus.com
[(2)] Airbus Defence & Space, 31 rue des cosmonautes, 31402 Toulouse, France Email:antoine.certain@airbus.com

ABSTRACT

Next generation space missions will require more capable computers in order to implement either advanced navigation and control algorithms needed to increase the spacecraft autonomy and agility or on the payload side with complex scientific payload data pre-processing algorithms. The advent of low-power multi- and many-cores processor architectures provides critical real-time embedded systems with an unprecedented performance level, opening the door to implement more intelligent systems. These architectures however, pressure the software development process, as applications must be parallelised in order to exploit the huge performance capabilities they offer.

The complexity of parallel programming has already been identified as a major challenge in general purpose computing, and it is now exacerbated in the critical embedded systems domain, due to the non-functional properties that these systems must fulfilled, e.g. functional safety and time predictability.

The majority of architectures incorporate parallel programming models in their software development kits with the objective of easing programmability and exploiting performance capabilities. These models provide an interface that entails a level of abstraction enabling to expose parallelism while hiding the complexity of the processor architecture. Parallelism is accomplished by the definition of independent execution units and synchronization mechanisms that guarantee the correct control- and data-flow.

The decision on what parallel programming model is used directly affects portability. While high level models can be compiled and executed on different architectures supporting the same model without modifying the application, low level APIs (e.g. Pthreads) may require some tuning before they can be ported to another machine.

Overall, parallel programming models are a fundamental element to exploit the huge performance capabilities offered by the newest parallel heterogeneous architectures.

The purpose of this study is to demonstrate the benefits of using one of the most well-known parallel programming models, i.e. OpenMP, for the development of parallel space applications, in terms of performance, programmability and portability.

Two main goals are identified:

- Improve overall system performance by exploiting the most advanced parallel embedded architectures targeting the space domain
- Improve the parallel programming productivity by reducing the initial development efforts of systems based on parallel architectures,

By selecting and porting two representative payload processing use cases to OpenMP parallel programming model, and evaluating their deployment on two high-end computing devices, GR740 in the radiation hardened components family and latest Kalray Coolidge MPPA as COTS, the project execution successfully demonstrated that usage of OpenMP parallel programming could facilitate the development, and analysis of parallel real-time space applications.

Development framework is completely based on open source solutions. Cross compiler for the selected targets is mainstream GNU GCC and includes OpenMP 4.5 runtime and open source observability tools provided by Barcelona Supercomputing Center (Paraver and Extrae) were ported from HPC mainstream to the selected hardware targets, and exercised through the selected software use cases.

1. PREPARATORY PHASE

The preparatory phased purpose was to refine and summarize the actual needs, analyse the potential software solutions and associated challenges to build a solid plan for prototyping and evaluation on relevant targets.

1.1. System Needs and perspective for use of OpenMP

There are typically two different categories of on-board data processing platforms in spacecraft. Some computers are responsible for spacecraft or instrument control; others are responsible for the on-board processing of instruments data, more often located on the Payload segment.

For both categories, we foresee that next generation space missions will require more capable computers, with respect to what we can implement today using proven technologies like the LEON processor. On the control side, spacecraft autonomy and agility need to be improved thanks to new sensors data acquisition and processing for in-situ real-time usage, such as for instance visual based navigation. On the payload side, scientific data pre-processing algorithms as well as more robotics for in-orbit/on-Planet operations are

foreseen with, in the long term, the foreseeable usage of artificial intelligence.

Two tracks are identified by Airbus Defence and space, largely shared with other European space Industrial for achieving such goal, and supported by the technology roadmap of the European Space Agency and national agencies:

- The development of new processing architectures on new rad-hard technologies such as, among others, the recent GR740 and HPDP with the STM-65nm and the future Dahlia/Brave-Ultra targeted for the new STM 28nm FD-SOI provide higher running frequencies with more efficient processing devices. On this technology track, developments of High performance computers are on-going which all require the capability to efficiently develop software on Multi-Core devices.
- The development of mitigation techniques enabling the use of commercial of the shelves processors (COTS) both as FPGA and micro-processors in the space environment. This way has been supported and matured through the ESA "COTS Based Computers" studies in which Airbus Defence and Space was strongly involved. For instance, this track which is more easily applicable for the LEO orbit has enabled the new Airbus product line "PureLine" based on automotive quality grade parts.

In this context, the emergence of complex computing architecture embedding many processing cores on single device is a real opportunity to drastically increase the on-board processing capability for both of these technology development tracks.

A wide range of devices are relevant for satisfying these needs:

- FPGAs (Field-Programmable Gate Arrays) use configurable logic blocks and implement a wide range of applications. They tend to outperform with streamed algorithms.
- DSPs (Digital Signal Processors) are optimized for single thread signal processing applications which mainly use typical signal processing operations. DSPs tend to perform these operations using fewer instructions than GPPs, and with lower power consumption. Some DSPs have a SIMD (Single Instruction, Multiple Data) architecture giving them Data Level Parallelism (DLP) capabilities.
- GP GPUs (General Purpose Graphics Processing Units) There is a wide variety of GPU architectures, but they usually consist of up to 32 cores, with each core containing up to hundreds of processing units. The cores have their own instruction stream and the same stream is shared by the processing units of a core. The cores have a private, local memory and they communicate through the GPU shared memory.
- Many-core processors are designed to reach high throughputs using Thread Level Parallelism (TLP). Usually, the cores are quite small in order to fit a high number of them on a chip with reasonable power consumption.
- Multi-core GPPs (General Purpose Processors) outperform in single-threaded applications and mostly take advantage of ILP (Instruction Level Parallelism) thanks to complex pipelined and superscalar architectures. This parallelization is implicit since it is automatically managed by the hardware. GPPs rely on cache memories to reduce the latency of memory transfers. Multi-core architectures make it possible to run some threads in parallel.

Some systems use a combination of devices to implement the overall application. To choose the most suitable hardware solution for a specific algorithm, several items such as cost, speed, flexibility, power and optimization as well as the design environment (team's skill, design tools, licensing…) should be taken into account.

The complexity of parallel programming has already been identified as a major challenge in general purpose computing, and it is now exacerbated in the critical embedded systems domain, due to the non-functional properties that these systems must fulfil, e.g. functional safety and time predictability.

OS's and RTOS's already provide all the services required to designing a parallel application. Nonetheless, reaching a high level of parallelism by only using the services of the OS is hard as it requires finding a good balance between the loads of the different cores with very rudimentary tools.

The majority of architectures incorporate parallel programming models in their software development kits with the objective of easing programmability and exploiting performance capabilities. These models provide an interface that entails a level of abstraction enabling to expose parallelism while hiding the complexity of both the processor architecture and the OS.

OpenMP presents the following advantages over its competitors:

- Different evaluations demonstrate that OpenMP delivers performance and efficiency tantamount to highly tunable models such as TBB, CUDA, OpenCL, and MPI. Regarding low-level libraries such as Pthreads, it offers multiples advantages such as: a) robustness without sacrificing performance, and b) OpenMP does not lock the software to a specific number of threads.
- The code can be compiled as a single-threaded application by either disabling support for OpenMP or assigned a single computing unit (i.e., OpenMP thread), thus easing debugging.

- OpenMP has a large and experienced community that has constantly reviewed and augmented the language for the past 20 years achieving great expressiveness. OpenMP is the de-facto shared-memory programming model standard. The language defines a very powerful tasking model that allows expressing fine-grained, both regular and irregular, and highly-dynamic task parallelism, augmented with features to express task dependencies. The latest specification of OpenMP also incorporates new features that facilitate the coupling of a main host processor to accelerator devices by defining both synchronous and asynchronous communication between them.
- OpenMP is widely implemented by several chip and compiler vendors (e.g. GNU, Intel, IBM, Kalray, Texas Instruments). Typically, ensuring functional correctness in languages that are not developed for such purpose is not straightforward. Nonetheless, OpenMP static correctness techniques are quite mature and simple compared to other parallel programming models which are not designed with correctness in mind (e.g. the low level Pthreads API or the Message Passing Interface, MPI).

Overall, OpenMP is a good candidate to be used in the real-time embedded domain in general, and in the space domain specifically by virtue of its benefits.

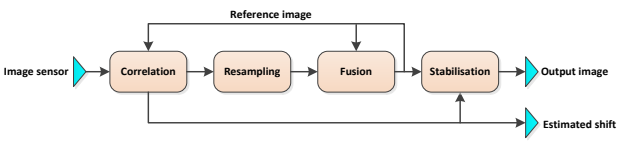
1.2. Evaluation Plan

Based on the definition of those requirements, a work plan for applicability of OpenMP on different targets was established, so as to answer the project goals objectives and provide crucial pieces of information amongst which:

- a measurable view of the performance of high demanding algorithms with two different multiple cores targets using an OpenMP implementation
- feedback and experience on the efficiency of OpenMP to parallelize application
- some practical view on the portability offered by the OpenMP framework

A set of criteria was defined and applied to select software and hardware to be exercised.

1.3. Use Case: HRGEO



The complete algorithm is divided into several computation stages. The sensor image is a RGB image with the three composites values for each pixel (no bayering is considered here).

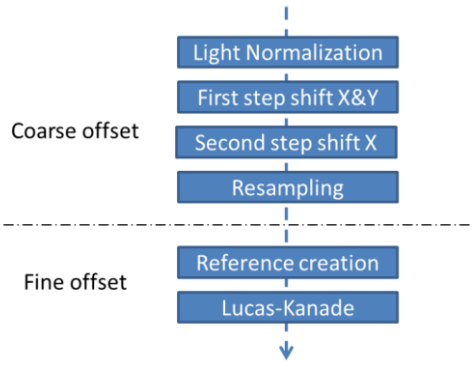
The algorithm includes image registration, resampling and fusion. The goal of the algorithm is to improve the S/N ratio on a static image by merging multiple images from a stream coming out of a more or less steady camera. The algorithm creates and manages a geometric model which fits the image displacement. It accumulates the successive images into a single reference image after compensating for the displacement with sub-pixel accuracy.

The implementation uses fixed-point representation for image correlation and keeps only a few operations in double precision.

1.4. Use Case: Adaptive Mirror

Future earth observation satellites with high resolution requirements will need primary mirrors of huge size. Due to their big size, they will suffer thermoelastic deformations. Adaptive optics allows fixing these defaults thanks to a deformable mirror actuated by a myriad of piezo actuators. The wave front defaults are measured on board by a Shack-Hartmann analyzer with its image sensor. Wavefront sensors aim at analyzing the shape of an incident beam's wavefront in order to identify aberrations caused by light traveling through individual optics or optical assemblies.

The algorithm is composed of two steps. It is applied on a 12x12 matrix of lenses for 3 detectors that are independent. But not all lenses must be processed, only the ones with a light intensity above a certain threshold. The light intensity is a constant, thus the number of lenses to be processed can be defined beforehand. 104 lenses have to be processed in the current implementation.



As illustrated in above, the image processing partition is composed of two parts: one that computes a first rough estimate of the shifts and interpolates the image with this shift; and another one that uses the interpolated image to compute a very precise shift.

1.5. Hardware Targets Selection

Based on a set of criteria not detailed in this public report on purpose, two hardware targets were selected amongst the following candidates { GR740, HPDP, RC64 } for the radiation hardened targets and { Zynq, Zynq MPSoC, Kalray Coolidge MPPA } for the COTS ones.

The GR740, the Coolidge and the Zynq 7000/UltraScale+ appeared to be the targets the best suited to the HP4S study. This is mostly because they implement hardware cache coherency mechanisms, they support SMP and their software stack is based on open-source tools. On the other hand, the other two targets that are considered (the RC64 and the HPDP) have software stacks based on proprietary tools and a high effort would be required to implement a reduced OpenMP runtime on these targets.
Final selected ones are GR740 and MPPA Coolidge.

2. IMPLEMENTATION PHASE

2.1. Use case software porting to OpenMP methodology

The implementation was divided into two stages for this study:

- Porting for first sequential execution on the targets.
- Parallelization of the applications with OpenMP (low optimization effort).

The sequential version of application serves as a reference for the considered evaluation metrics, such as memory occupation and of course execution time performance speed up with the parallelized version.
A single parallel implementation is performed for each use-case and the efficiency of this implementation is evaluated on the different targets to evaluate portability capabilities of OpenMP.
The methodology followed to parallelize the use-cases is presented below:

2.2. Identifying the hotspots of the algorithm

The hotspots can be identified by theoretical analysis of the algorithm or empirically, by profiling the sequential program. Initial measurements and profiling executed on x86_64 benefited from standard HPC tools through GNU gprof x86_64 and Intel VTune Analyzer.
This allowed selecting the main functions to parallelize for each use case, yielding 99.8% of the sequential execution time for HRGEO application and 90% for the adaptive mirror one.

2.3. Choosing an appropriate parallelization strategy for each hotspot

Selecting a parallelization strategy consists in identifying the appropriate type of parallelism and the appropriate OpenMP constructs to implement it. With a good parallelization strategy, it is possible to achieve high performance with minor modifications to the serial code.
The different types of parallelism are:

- Data parallelism, where each thread executes the same task on different sets of data.
- Task parallelism, where each thread executes a different task. This can be used to parallelize recursive algorithms or to create pipelines.

Both data and task parallelism can be implemented with different OpenMP constructs:

- Worksharing constructs: The work is divided into work items which are distributed over a team of threads, either statically or dynamically.
- Tasking constructs: Although writing task-parallel programs with worksharing constructs is possible, it is often inconvenient. Tasking constructs have been introduced to provide a convenient syntax and more flexibility for task parallelism.

To summarize, worksharing constructs are suited to data parallelism (do/for loop constructs) while tasking constructs implement task and data parallelism more conveniently and with more flexibility.
The synchronization and data management clauses are applied to the different worksharing and tasking constructs. Synchronization points can also be added at specific points of the program.

2.4. Investigating the potential issues when the targeted acceleration is not met

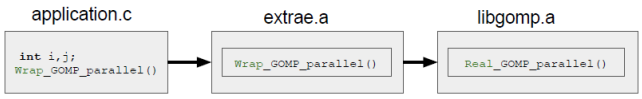
The following points can be responsible for low performances:

- Poor memory accesses due to an inefficient use of cache memory. To maximize the cache efficiency, each thread should be able to work on independent sets of data and the cache spatial and temporal locality principles should be taken into account. Achieving this may require source code modifications.
- Overhead due to synchronization, scheduling or context switching. Several solutions not detailed here can be considered depending on the origin of the overhead.
- Unbalanced workload across threads. Several solutions not detailed here can be considered depending on the origin of the unbalance.

2.5. Porting observability tools

Compilation tool chain already support OpenMP 4.5 on both targets through RCC 1.3-rc6 and ACE SDK 4.1.0 respectively on GR740 and Kalray Coolidge targets.
There are two main additional tools used during this evaluation: Extrae which is used to instrument code and Paraver which displays the data collected by Extrae. Both have been developed and are maintained as open-source project by the Barcelona Supercomputing Centre (BSC). Extrae is a rich tool which takes advantages of reusing packages from others developers' community.
It supports natively CUDA, OpenCL and OpenMP environments. In its basic version it allows to measure time and the number of instructions executed. To get memory usage information (which could be very interesting for optimization) then the PAPI library should be used. PAPI provides a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors but the

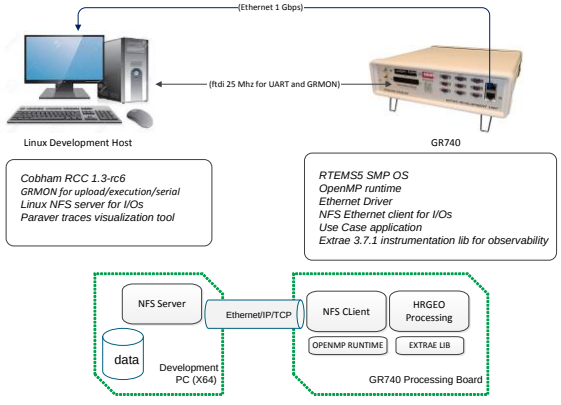
usage of this library requires a lot of re-compilation, including the kernel.
Extrae uses an XML configuration file at runtime. This configuration file is used to enable or disable some profiling features, and to set some parameters. Once the program has been executed, the profiling data can be displayed and analysed with Paraver.
Extrae adaptation to the embedded targets consisted in replacing some of the techniques applied on regular HPC context to comply with the embedded constraints.
Ported extrae intends to keep benefiting from all the features of the HPC mainstream version; however ports realized during this study present some limitations due to their alpha version and bounded effort. One of the main changes is related to automatic instrumentation. Originally based on the LD_PRELOAD dynamic calls interception at runtime, it had to be adapted for embedded targets using symbol wrapping at compile time using linker flags, as illustrated in following Figure.



Another modification lies in the hardware counter gathering process originally based on PAPI library. While PAPI library is natively available on Kalray Coolidge MPPA, hardware counters function was enhanced to also benefit from GR740 L4STAT counters, through L4STAT driver.

3. Evaluation Phase

3.1. GR740 Test setup



As illustrated in Figure above, the test setup is mainly composed of a GR-CPCI-GR740 Quad-Core LEON4FT Development Board, connected to a Linux host development machine.

I/Os consist of input data file either directly linked with the application or retrieved from development host through NFS over a 1Gbps Ethernet link. This NFS share is also used to retrieve both functional and instrumentation outputs.
The software stack on the development host is composed of RCC 1.3-rc6, grmon Pro 2.0.98, nfs-kernel-service, BSC Paraver, version 4.8.1.
The software stack on the embedded platform is composed of RTEMS 5 SMP, Actual Use Case algorithm linked, OpenMP 4.5 runtime, Extrae 3.7.1 instrumentation library from BSC ported to GR740

3.2. KALRAY TEST SETUP

The Kalray test setup is based on the MPPA® DEV4 workstation which is a complete X86 – MPPA® based environment, packaged with Kalray S/W, for easy benchmarking and development of accelerated systems.

This development is composed of a x86_64 host part running a standard Linux distribution and a KONIC200 PCIe Programmable accelerator card, hosting the Coolidge MPPA device.
Two modes of operation were exploited during the study. The first one is the JTAG mode on single cluster. It consists on the generation of a standalone executable intended to be run and parallelized on a single cluster on top of the Kalray Cluster OS. Deployment of the executable on target relies a JTAG link with PCIe acceleration for executable or binary blobs transfer to/from MPPA® Coolidge target. The JTAG mode also offer semi-hosting capabilities allowing to access host file system through regular file manipulation calls, with inherent
Second mode of operation explored is OpenCL offloading where the Coolidge manycore is considered as an accelerator. The kernels that will run in the accelerator must be compiled in isolation with the OpenCL compiler, and then linked with the host application part. All the communication is done through PCIe.
The software stack on the host platform is composed of Kalray SDK ACE 4.1.0, based on gcc 7.5.0, and same x86 4.8.1 Paraver version than the one used for GR740.
On the embedded part the stack is composed of ClusterOS, which is a light OS optimized for MPPA® cluster, with openMP 4.5 supports and multithreading

support through pThread POSIX, as well as a port of Extrae 3.7.1 to the Coolidge architecture.

3.3. Test Scenario

Test scenario for both HRGEO and MIRROR use cases execution on target follows an incremental approach.
First the algorithm is executed in its pure sequential form to obtain a reference baseline.
Then algorithm is executed with OpenMP runtime integrated but only 1 core mainly to measure open MP overhead in terms executable size overhead and execution time overhead.
Then the number of exploited cores is incrementally increased to measure actual multicore performance gain thanks to parallelization.
Finally OpenMP Extrae is integrated on the scenario with the maximum number of cores active to check the observability mechanism and verify proper and efficient parallelization of the code.
For each of those tests functional correctness of parallelized code is verified by comparing outputs against expected ones for the predefined set of inputs.

3.4. Quantitative metrics

The figure of merit is used to detect an unbalanced workload across threads for parallelized software sections:

$$FOM = \frac{\sum task\_duration}{total\_duration}$$

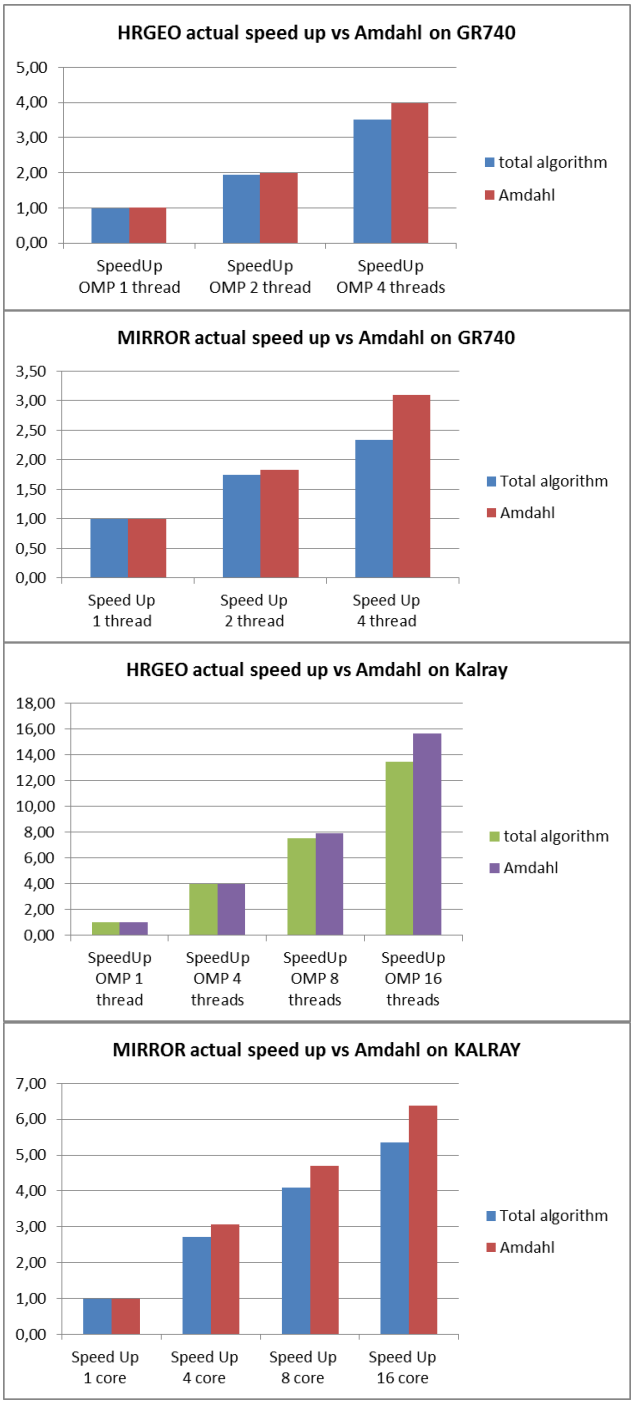
The FOM is 1 for a single core processor because all tasks are executed sequentially. The FOM is N for a perfectly balanced workload across N cores
The theoretical total speed-up achieved by parallelizing the hotspots is provided by Amdhal's criteria and it has to be taken into account when selecting the hotspots to be parallelized.

$$S_N = \frac{1}{\frac{F_{parallel}}{N} + F_{sequential}}$$

- S_N is the theoretical speed-up of the whole program with N cores.
- F_parallel is the fraction of time consumed by the regions considered for parallelization.
- F_sequential is the fraction of time consumed by the sequential regions

3.5. Speed Up

This section presents the measure figure of merit for each individual parallelized loop as well as the overall total algorithm time speed up, including remaining sequential phases, to be compared to theoretical execution time obtained with Amdahl's law.



3.6. Functional Correctness

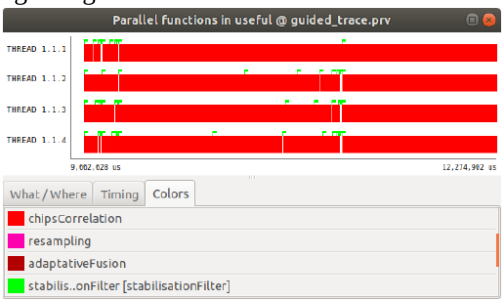
All tests presented correct outputs, checked based on md5sum signature.
However when trying to optimize total algorithm parallelization speed up of the adaptive mirror application by trying to parallelize the buildRef function which was remaining sequential in our port, we faced an interesting issue worth to be discussed. Parallelization of this function actually impacts the parallelization of a floating point summation. While commutativity

property holds for a real numbers summation, it is known not to be necessarily the case with floating points and in our case precision of the final result is indeed impacted by the order in which this sum operation is performed.
Parallel code results in a different order of operation than the sequential version. This forbids us to get a bit accurate result between sequential and parallelized version of the algorithm. In our specific case as the unique success criteria was a bit accurate final image result compared to the sequential execution we did not focus any additional efforts on analyzing the actual discrepancies and evaluating if the non-accurate results were actually still acceptable in terms of result overall quality.

3.7. Qualitative metrics on Extrae exploitation

During the project we evaluated the benefit of Extrae instrumentation and Paraver visualization to provide better visibility and understanding of parallel execution of the algorithm on actual target, providing crucial pieces of information in the efficient usage of the multicore resources.
During HRGEO initial parallelization no fine tuning was used and static scheduling was applied, resulting in unbalanced execution illustrated below where fourth core starves.

Analysis of the last loop code shows that it traverses a really small iteration space and static partitioning cannot divide the space perfectly into 4 threads, so the last one gets less workload. The processing consists of two nested loops and only outer one is actually parallel. As the loops are perfectly nested they can be collapsed so as to increase the iteration space and better balance the chunks split.
Finally as some threads do not get enough work in the first loop, a switch to guided schedule could help by forcing a mode dynamic scheduling with big chunks at the beginning then small at the end.



4. SUCCES CRITERIA

4.1. Definition

A set of success criteria were defined and detailed hereafter:
(Criteria#1) OpenMP runtime execution time overhead
Acceptable bound is set to 5% for a monocore usage of OpenMP
(Criteria#2) OpenMP runtime code / data size overhead
Acceptable bound is set to 500kB of code and same for data.
(Criteria#3) Extrae embedded instrumentation library timing overhead
Acceptable bound is set to 5%
(Criteria#4) Extrae timing measurement accuracy
Here the accuracy must be in the order of magnitude of what is achieved with manual instrumentation through available hardware timers
(Criteria#5) Multicore achieved speed up
Up to 8 cores : [Amdahl-1; Amdahl]
From 16 cores considering non-optimized usage of Kalray cluster (application linked entirely in DDR) we relax a bit the target on purpose: [Amdahl-2; Amdahl]
(Criteria#7) Functional Correctness of parallelized code Bit accurate output image
(Criteria#6) Multicore efficient exploitation, explored with Extrae instrumentation
(Criteria#8) Initial Instrumentation Effort
(Criteria#9) Porting effort when switching to a new target
(Criteria#10) Observability capabilities through instrumentation

4.2. Evaluation

Success Criteria	GR740	KALRAY	TARGET
(Criteria#1) OpenMP runtime execution time overhead	1%	1%	< 5%
(Criteria#2) OpenMP runtime size overhead	~100 KB	~100KB	< 500KB
(Criteria#3) Extrae library timing overhead	3%	2%	<5%
(Criteria#4) Extrae timing accuracy	<µs	<µs	<µs
(Criteria#5) Multicore achieved speed up	*Hrgeo:* **3.52** [2.99-3.99] *Mirror:* **2.34** [2.10-3.10]	*Hrgeo:* **7.51** [6.92-7.92] **13.42** [13.35-15.65] *Mirror:* **4.09** [3.69-4.69] **5.36** [4.38-6.38]	*cf. [x-y]*
(Criteria#6) Multicore efficient exploitation, explored with Extrae instrumentation	PASSED	PASSED	
(Criteria#7) Functional Correctness	bit accurate output	bit accurate output	
(Criteria#8) Initial Instrumentation Effort	PASSED	PASSED	
(Criteria#9) Porting effort to a new target	PASSED	PASSED	
(Criteria#10) Observability	PASSED	PASSED	

	Detailed success evaluation
(Criteria#1)	Execution time overhead is absolutely acceptable in the order of magnitude of a few percents on single core execution, and only one blocking point was found on Mirror use case when trying to parallelize one of the low overall execution time contributors.
(Criteria#2)	Executable size overhead is in the order of 5% for the considered use case which are not that big. Absolute overhead size is in the order of magnitude of a hundred of kbytes
(Criteria#3)	Extrae execution time overhead is very low when used in monocore mode
(Criteria#4)	Extrae measurement is as accurate as the available Performance Monitor or core timers can be. So as precise as what would be added through manual source instrumentation, with the advantage of not modifying the source code for those openMP runtime calls that can be automatically trapped thanks to wrapping
(Criteria#5)	Achieved speed up is very good, following Amdahl's considering the amount proportion of parallelized function. Instrumentation provides a convenient way to retrieve performance counters, and internal insights, in order to help profiling and support investigation of the areas with limited speedup.
(Criteria#6)	Load balancing amongst core is quite easy to monitor thanks to Extrae instrumentation, allowing to tune parallelization annotations appropriately
(Criteria#7)	No errors were detected during tests: expected functional behavior with parallelized version of the algorithm was preserved.
(Criteria#8)	Initial parallelization effort was quite low, limited to a few pragma added to the code. Most of the coding effort was dedicated to wrapping of the algorithm to the target, operating system integration and configuration, and I/O management, rather than actual parallelization.
(Criteria#9)	Annotations or functional algorithm code was not modified from a target to another, only specific target init or I/O management functions.
(Criteria#10)	Extrae provides good automatic observables, user friendly way to defined additional manual events, and assistance for automatic accurate Hardware Counters retrieval.

5. CONCLUSION

Both selected targets SDKs were already supporting a mature openMP runtime, demonstrating the ecosystem is ready on both for radiation hardened and COTS components perspective. Porting based on a GNU GCC mainstream compiler is completely in line with current state of the art and golden rules applied in the scope of embedded software development as far as compiler choice is concerned..

The initial parallelization effort of legacy code revealed to be low, as expected, thanks to non-intrusive openMP annotations scheme. Selected annotations used to reach the decent performance improvement presented in this report were very limited in number and complexity, and selected so as to preserve determinism from one execution to another. An opposite choice could have been taken so as to introduce some more random dynamic scheduling schemes to capitalize on complexity to avoid deterministic worst cases.

The porting effort when switching from the first radiation hardened hardware target to the second COTS other proved to be inexistent as openMP annotations remained strictly unchanged. This is applicable as well to Extrae activation and manual instrumentations. The only customization consisted in openMP environment, .i.e. number of available cores.

Open source Extrae observability library and its associated Paraver visualization tool both provided by BSC and so far mainly targeting HPC mainstream world, were successfully ported to GR740 and MPPA Coolidge ManyCore. They provided all the expected features to sustain efficient profiling, verification and parallelization tuning. The tooling required a reasonable learning phase, is well documented and offers a standardized hardware agnostic interface allowing focusing on the actual data providing added value to the final end user mainly in the form of graphical intuitive load dispatch representations, timing information, and hardware counters. Such knowledge is mandatory to ensure efficient usage of multi and many cores, and support production delay shortening.

However it is important to remind that despite we demonstrated during this evaluation phase that the selected algorithms can be efficiently parallelized and observed, this is only valid for those two selected representative algorithms, and of course cannot prove applicable to any legacy code. We also found some limitation in the strategy for some of the functions where the ideal speed up cannot be reached either due to openMP overhead or due to actual hardware resource usage bottlenecks. While mitigation of the detected poor IPCs performance in well balanced parallel phase remain to be defined, we however confirmed that all possible observables would be available through the evaluated OpenMP framework to detect such corner cases and support such analysis.

As a result this evaluation comforts the idea that OpenMP could and should be seriously considered in the scope of future R&D multicore roadmap, as well as for rapid prototyping in advanced studies, especially when considering new space approaches.

As stated before HP4S defined two strategic end goals:

G1. Improve overall system performance. Effectively master and exploit the most advanced parallel embedded architectures targeting the space domain.

G2. Improve the parallel programming productivity. Reduce the development efforts of systems based on parallel architectures, while fulfilling system's functional and non-functional (time predictability) requirements

Those two goals derived in a set of technical measurable objectives, listed hereafter:

O1. Facilitate the development, timing analysis and execution of parallel real-time space applications using the OpenMP parallel programming model.

O2. Evaluate the interest and porting effort of a list of homogeneous and heterogeneous foreseen COTS and RadHard hardware targets in the space domain with OpenMP programming model and framework.

O3. Adapt the OpenMP runtime libraries to ensure that the timing guarantees devised at analysis time can be guaranteed at deployment time.

O4. Evaluate state-of-the-art compiler techniques to guarantee that parallel OpenMP

O5. Demonstrate the portability benefits of the OpenMP parallel programming model.

The outcomes of the previous preparatory and implementation phase, completed by the experiments results obtained during the evaluation phase and presented in the report allow us to state that O1, O2, and O5 are achieved while O3 and O4 remain open for future work.